MANUAL DE PROGRAMACIÓN I

4. UNIDAD 4: ESTRUCTURAS

4.1. ESTRUCTURA SELECTIVA

De acuerdo con las condiciones de un problema en ocasiones se necesita tener la posibilidad de controlar cuales son las instrucciones que se deben ejecutar y en qué momento, para esto los compiladores traen instrucciones que le permiten al programador elegir entre una o varias alternativas.

4.1.1. CONDICIONES

Las condicionales son expresiones booleanas, que se emplean para controlar la secuencia en que deben ejecutarse los pasos de un algoritmo. Generan un valor lógico *verdadero* (1) o *falso* (0), esto es, pueden tomar dos valores: *verdadero* (*true*) y *falso* (*false*).

Las instrucciones condicionales controlan si una instrucción o lista de instrucciones se ejecutan en función del cumplimiento o no, de una condición.

Esta instrucción se denomina **si** y tiene la siguiente forma:

si *(condición)*
    entonces *instrucciones*
sino
    instrucciones
fin_si

Si el valor generado por la condición es *verdadero* se ejecutan las instrucciones que siguen a la palabra ***entonces***. Si el valor generado por la condición es *falso*, se ejecutan las instrucciones que siguen a la palabra ***sino***, en caso de que no se requiera especificar instrucciones por *falso*, simplemente se omite esta parte y las instrucciones por *verdadero* se ejecutarían:

si *(condición)*
    entonces *instrucciones*
fin_si

Ejemplo 1: Leer dos variables y compararlas así:

Variables: a, b

```
1  Algoritmo Ejercicio1
2      Escribir "Ingrese el primer numero: "
3      Leer a
4      Escribir "Ingrese el segundo numero: "
5      Leer b
6      Si a<b Entonces
7          a<-a+1
8      Fin Si
9      Escribir "El primer numero es: ",a
10     Escribir "El segundo numero es: ",b
11 FinAlgoritmo
```

donde:

- ✓ La expresión booleana es (*a* < *b*). Línea 6
- ✓ Si el valor generado es *verdadero* (por ejemplo, si *a=2* y *b=3*) entonces se ejecuta la instrucción "asignar a, *a* el valor de *a + 1"*
  a ← *a* + *1*. Línea 7
- ✓ En caso de que la expresión booleana sea *falsa* (por ejemplo, *a=3* y *b=2*) entonces la secuencia de instrucción a ejecutar seria la siguiente al ***fin_si***, es decir, *escribir el valor de a, b.* Líneas 9 y 10.

Ejemplo 2: Leer dos variables y compararlas así:

Variables: a, b

```
1  Algoritmo Ejercicio2
2      Escribir "Ingrese el primer numero: "
3      Leer a
4      Escribir "Ingrese el segundo numero: "
5      Leer b
6      Si a<b Entonces
7          a<-a+1
8      SiNo
9          b<-b+1
10     Fin Si
11     Escribir "El primer numero es: ",a
12     Escribir "El segundo numero es: ",b
13 FinAlgoritmo
```

donde:

- ✓ La expresión booleana es (a < *b*). Línea 6
- ✓ Si el valor generado es *verdadero* (por ejemplo, si *a=2* y *b=3*) entonces se ejecuta la instrucción "asignar a, *a* el valor de *a + 1"*
  a ← *a* + *1.* Línea 7
- ✓ A diferencia del anterior, en caso de que la expresión booleana sea *falsa* (por ejemplo, *a=3* y *b=2*) entonces se ejecuta la instrucción "asignar a, *b* el valor de *b + 1"*
  b ← *b* + *1*. Línea 9
- ✓ Por último, la secuencia de instrucción a ejecutar seria
  Escribir el valor de a, b. Líneas 11 y 12

4.1.2. CLASES DE CONDICIONES

Las condiciones son expresiones de comparación y para crearlas, se usan los operadores relacionales así:

a = b	*a es igual a b*
a != b	*a es diferente de b*
a < b	*a es menor que b*
a <= b	*a es menor o igual que b*
a > b	*a es mayor que b*
a >= b	*a es mayor o igual que b*

Las *Condiciones Simples* son condiciones sencillas que establecen una relación entre: Variable-Variable, Variable-Constante, Variable-Expresión o Expresión-Expresión, por ejemplo:

a >= 7 *x != y*

Las *Condiciones Compuestas* permiten enlazar *Condiciones Simples* para formar otras más complejas; las condiciones son enlazadas entre sí por medio de los *Operadores Lógicos* (Y – O), por ejemplo:

a > b **y** *a > c*
x != y **o** *x = z*
no (a > b) **y** *no (x = y)*

Ejemplo 3: Leer una variable y compararla así:

Variables: a, x

```
1  Algoritmo Ejemplo3
2      Escribir "Ingrese un numero: "
3      Leer a
4      Si a<10 o a>20 Entonces
5          x<-3
6      SiNo
7          x<-0
8      Fin Si
9      Escribir "El valor de x es: ",x
10 FinAlgoritmo
```

donde:

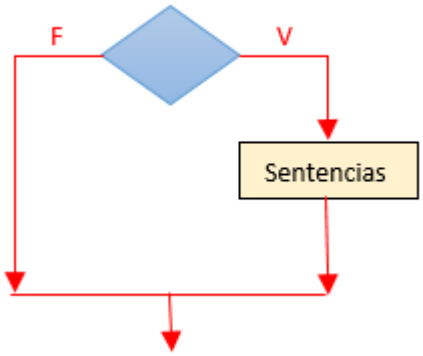
- ✓ Si la condición es *verdadera*, (por ejemplo, si *a=3*) entonces se asigna a *x* el valor de *3*
  x = 3. Línea 5
- ✓ Si la condición es *falsa*, (por ejemplo, si *a=15*) entonces se asigna a *x* el valor de *0*
  x = 0 Línea 7

4.1.3. REPRESENTACIÓN DE ESTRUCTURAS CONDICIONALES

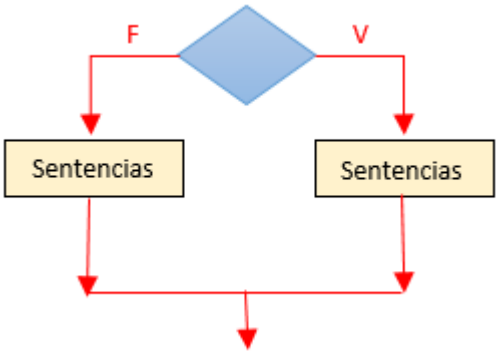
- ✓ **Usando Diagramas de Flujo**

Para representar un condicional se utiliza un rombo:

Si no se requiere especificar instrucciones por falso



Si se requiere especificar instrucciones por falso y verdadero:



4.1.4. CONDICIONALES SENCILLOS

Los condicionales sencillos son aquellos que solamente tienen una alternativa de salida:

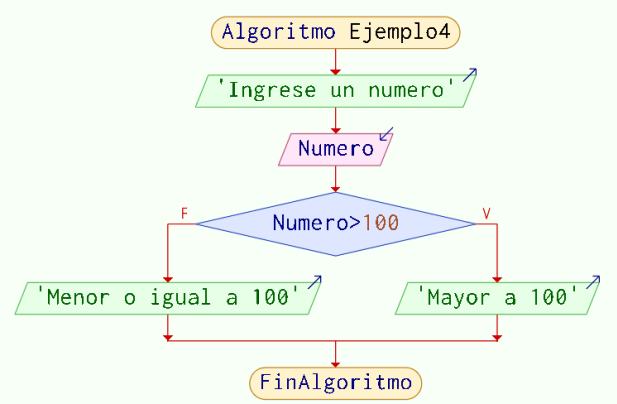
Ejemplo 4: Leer un número y escribir si es "mayor" o "menor o igual" que 100

Variable: Numero

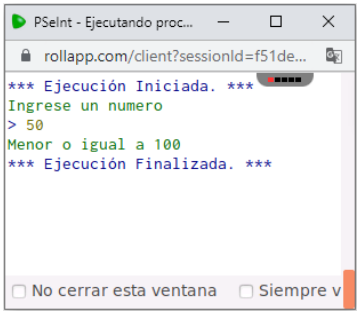
Pseudocódigo

```
Algoritmo Ejemplo4
    Escribir "Ingrese un numero"
    Leer Numero
    Si Numero>100 Entonces
        Escribir "Mayor a 100"
    SiNo
        Escribir "Menor o igual a 100"
    Fin Si
FinAlgoritmo
```

Diagrama de Flujo



Ejecución



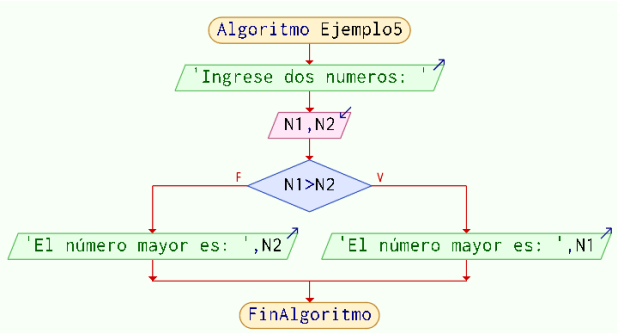
Ejemplo 5: Leer dos números, y determinar cuál es el mayor

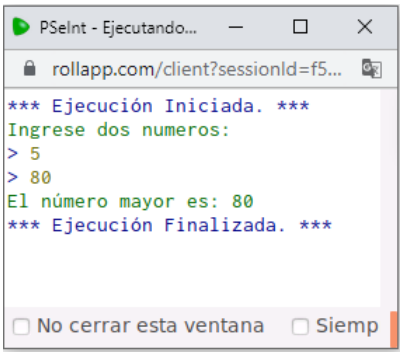
Variables: N1, N2

Pseudocódigo

```
Algoritmo Ejemplo5
    Escribir "Ingrese dos numeros: "
    Leer N1,N2
    Si N1 > N2 Entonces
        Escribir "El número mayor es: ",N1
    SiNo
        Escribir "El número mayor es: ",N2
    Fin Si
FinAlgoritmo
```

Diagrama de Flujo





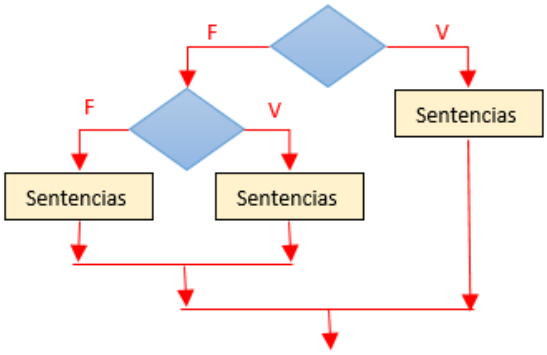
4.1.5. CONDICIONALES ANIDADOS

En ocasiones se necesita tener la posibilidad de controlar las instrucciones que se deben ejecutar entre más de una alternativa, para cumplir más de una condición. La instrucción *SI*, puede contener otras instrucciones *SI* anidadas.

La instrucción *SI* anidada por falso, tiene la siguiente forma:

si *(condición 1)*
    entonces *instrucciones*
sino
    si *(condición 2)*
        entonces *instrucciones*
    sino
        instrucciones
    fin_si
fin_si

En diagrama de flujo se presenta así:



En la forma anterior, se tienen dos condiciones para tres posibilidades de instrucciones. La negación de la primera condición resulta ser el segundo ***si***, el cual permite evaluar las dos posibilidades de la segunda condición.

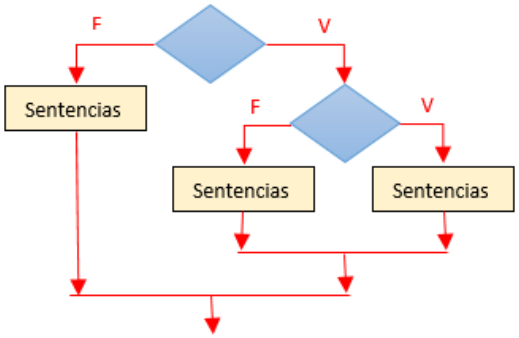
La instrucción *SI* anidada por verdadero, tiene la siguiente forma:

si *(condición 1)*
    entonces
    si *(condición 2)*

```
        entonces instrucciones
    sino
        instrucciones
    fin_si
sino
    instrucciones
fin_si
```

En este caso, la afirmación de la primera condición resulta ser el segundo ***si***, el cual permite evaluar las dos posibilidades de la segunda condición.

En diagrama de flujo se presenta así:



Se puede anidar un número indeterminado de instrucciones *SI*. de la habilidad del programador para controlar todas las posibilidades depende la eficiencia de esta instrucción.

Un ejemplo de cuatro condicionales es:

```
si (condición 1)
    entonces instrucciones
sino
    si (condición 2)
        entonces instrucciones
    sino
        si (condición 3)
            entonces instrucciones
        sino
            si (condición 4)
                entonces instrucciones
            sino
                instrucciones
            fin_si
        fin_si
    fin_si
fin_si
```

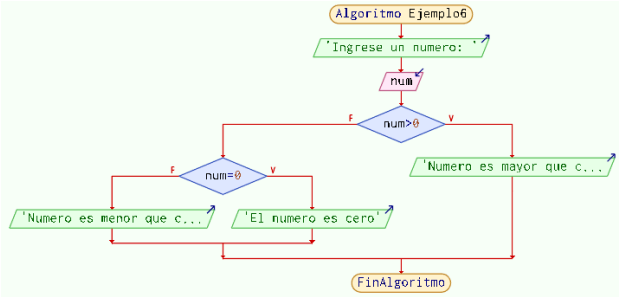
Ejemplo 6: Leer un número y escribir si es menor que cero, igual a cero o mayor que cero.

Variable: num

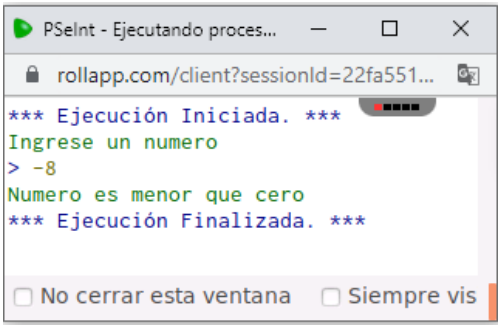
Pseudocódigo

```
Algoritmo Ejemplo6
    Escribir "Ingrese un numero: "
    Leer num
    Si num > 0 Entonces
        Escribir "Numero es mayor que cero"
    SiNo
        Si num = 0 Entonces
            Escribir "El numero es cero"
        SiNo
            Escribir "Numero es menor que cero"
        Fin Si
    Fin Si
FinAlgoritmo
```

Diagrama de Flujo



Ejecución



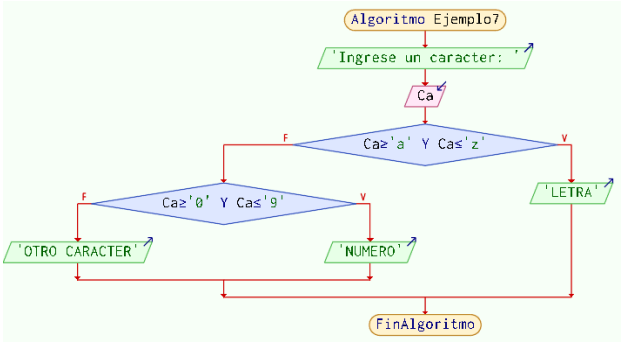
Ejemplo 7: Leer un carácter, determinar si es una letra, un número, otro carácter.

Variables: Ca - Carácter.

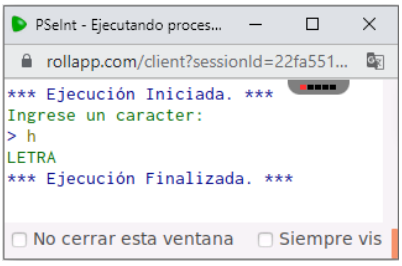
Pseudocódigo

```
Algoritmo Ejemplo7
    Escribir "Ingrese un caracter: "
    Leer Ca
    Si Ca ≥ 'a' y Ca ≤ 'z' Entonces
        Escribir "LETRA"
    SiNo
        Si Ca≥ '0' y Ca ≤ '9' Entonces
            Escribir "NUMERO"
        SiNo
            Escribir "OTRO CARACTER"
        Fin Si
    Fin Si
FinAlgoritmo
```

Diagrama de Flujo



Ejecución



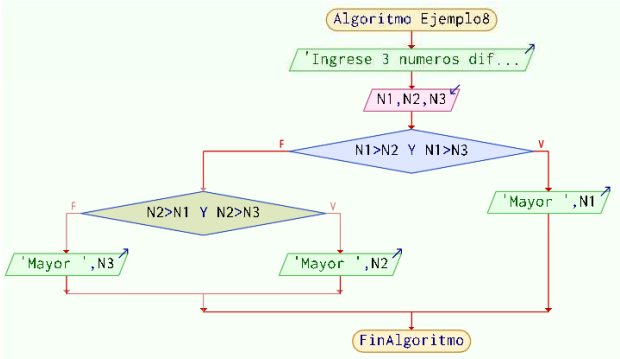
Ejemplo 8: Leer tres números diferentes, determinar cuál es el mayor.

Variables: N1, N2, N3

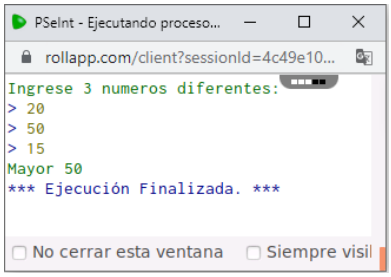
Pseudocódigo

```
Algoritmo Ejemplo8
    Escribir 'Ingrese 3 numeros diferentes: '
    Leer N1,N2,N3
    Si N1>N2 Y N1>N3 Entonces
        Escribir 'Mayor ',N1
    SiNo
        Si N2>N1 Y N2>N3 Entonces
            Escribir 'Mayor ',N2
        SiNo
            Escribir 'Mayor ',N3
        FinSi
    FinSi
FinAlgoritmo
```

Diagrama de Flujo



Ejecución



4.2. ESTRUCTURAS REPETITIVAS

Las *Estructuras Repetitivas* son aquellas que permiten que el computador ejecute varias veces un grupo de instrucciones, sin tener que volverlas a escribir. Estas instrucciones son *mientras que* y *para*.

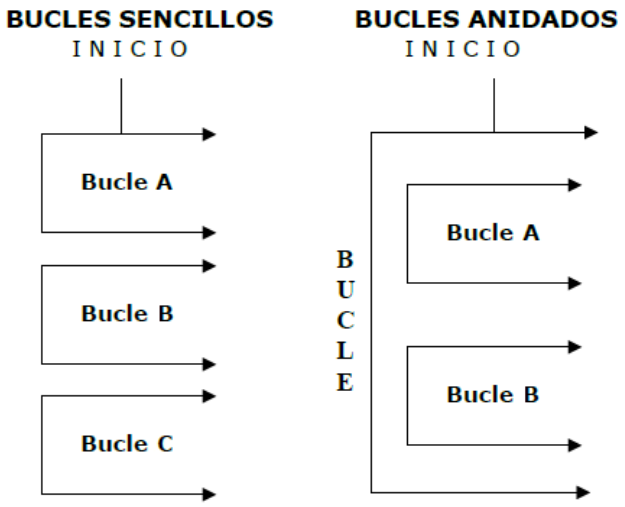
4.2.1. BUCLES E ITERACIONES

Un ***bucle*** o ***ciclo*** es una parte de algoritmo, cuyas instrucciones se repiten un número determinado de veces mientras se cumple una determinada condición.

Un ***bucle*** consta de tres partes fundamentales: *Decisión* (simple o compuesta), *Cuerpo del Bucle* y la *Salida del Bucle*.

Una ***iteración*** es la ejecución de todas las instrucciones del ciclo.

Los bucles pueden ser sencillos o anidados (ciclos dentro de otros ciclos).



4.2.2. CONTADORES Y ACUMULADORES

Los procesos repetitivos son la base del uso del computador. En estos procesos se necesita normalmente contar los sucesos o acciones internas del bucle. Una forma de controlar un bucle es mediante un *contador*.

- ✓ **Contador** es una variable cuyo valor se incrementa o decrementa en una cantidad constante por cada iteración.

  Contador ← *Contador* + *Constante*

  Ejemplo:

  Z ← *Z* + *2,*

  donde Z se incrementa en 2 en cada iteración

- ✓ **Acumulador** es una variable cuya misión es almacenar por cada iteración, cantidades variables resultantes de sumas o productos sucesivos.

  Acumulador ← *Acumulador* + *Variable*

  Ejemplo:

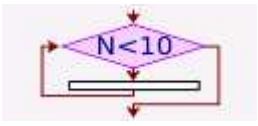
  S ← *S* + *N,*

  donde N se incrementa en una Variable

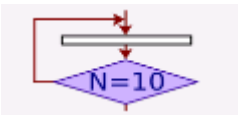
4.2.3. REPRESENTACIÓN DE ESTRUCTURAS REPETITIVAS

- ✓ **Usando Diagramas de Flujo**

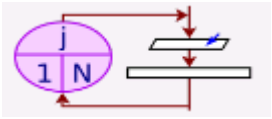
Una estructura repetitiva se representa de la siguiente manera:



Mientras



Repetir



Para

4.2.4. INSTRUCCIÓN MIENTRAS QUE

La instrucción *mientras*, nos permite ejecutar una instrucción o un grupo de instrucciones mientras se cumple una condición. Esta instrucción es utilizada, cuando se sabe o no se sabe el número de veces que va a ocurrir el ciclo.

La representamos en pseudocódigo de la siguiente forma:

mientras *(condición)*
hacer *instrucciones*
fin_mientras

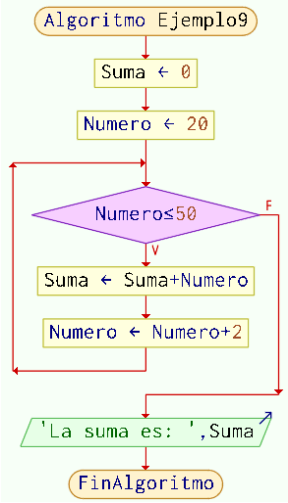
Ejemplo 9: Sumar los números pares comprendidos entre 20 y 50

Variables: Suma (Acumulador), Numero

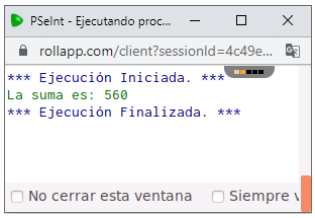
Pseudocódigo

```
Algoritmo Ejemplo9
    Suma <- 0
    Numero <- 20
    Mientras Numero ≤ 50 Hacer
        Suma <- Suma + Numero
        Numero <- Numero + 2
    Fin Mientras
    Escribir 'La suma es: ', Suma
FinAlgoritmo
```

Diagrama de Flujo



Ejecución



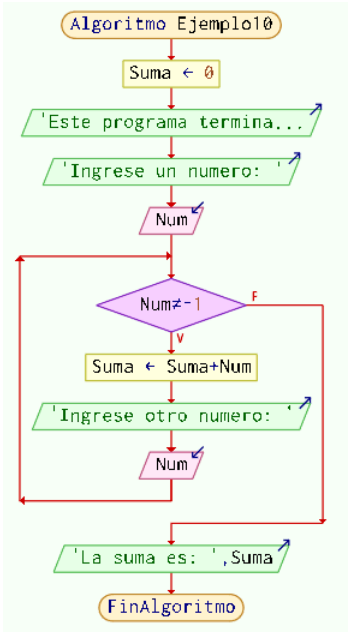
Ejemplo 10: Sumar una serie de números, el programa solicita los números por teclado hasta que se digite el numero -1

Variables: Suma (Acumulador), Num

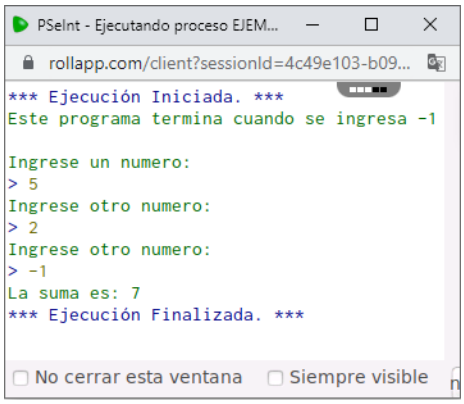
Pseudocódigo

```
Algoritmo Ejemplo10
    Suma <- 0
    Escribir 'Este programa termina cuando se ingresa -1'
    Escribir 'Ingrese un numero: '
    Leer Num
    Mientras Num≠ -1 Hacer
        Suma <- Suma + Num
        Escribir 'Ingrese otro numero: '
        Leer Num
    Fin Mientras
    Escribir 'La suma es: ', Suma
FinAlgoritmo
```

Diagrama de Flujo



Ejecución



4.2.5. INSTRUCCIÓN REPETIR

La instrucción *repetir*, nos permite ejecutar una instrucción o un grupo de instrucciones hasta que cumple una condición. La condición se valida al final no al inicio como es el caso de la instrucción *mientras*. Esta instrucción también es utilizada, cuando no se sabe el número de veces que va a ocurrir el ciclo.

La representamos en pseudocódigo de la siguiente forma:

repetir
    hacer *instrucciones*
hasta que*(condición)*

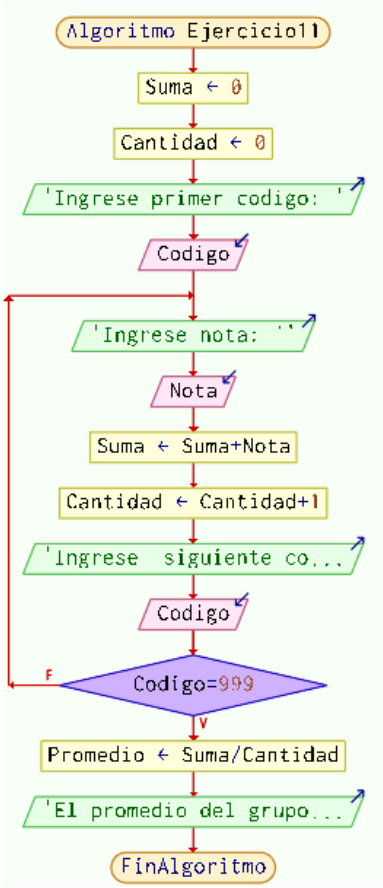
Ejemplo 11: Calcular el promedio de las notas de un grupo de alumnos. El programa finaliza el ingreso de notas cuando se digite el código 999.

Variables: Nota, Suma, Codigo, Cantidad, Promedio

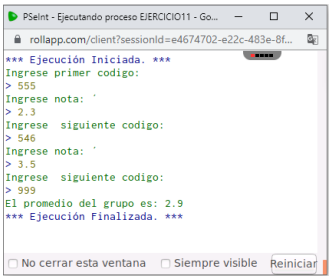
Pseudocódigo

```
Algoritmo Ejercicio11
    Suma <- 0
    Cantidad <- 0
    Escribir 'Ingrese primer codigo: '
    Leer Codigo
    Repetir
        Escribir 'Ingrese nota: ´'
        Leer Nota
        Suma <- Suma + Nota
        Cantidad <- Cantidad + 1
        Escribir 'Ingrese  siguiente codigo: '
        Leer Codigo
    Hasta Que Codigo=999
    Promedio <- Suma / Cantidad
    Escribir 'El promedio del grupo es: ',Promedio
FinAlgoritmo
```

Diagrama de Bloques



Ejecución



4.2.6. INSTRUCCIÓN PARA

La instrucción para permite repetir una instrucción o un grupo de instrucciones un número determinado de veces. Esta instrucción solo es utilizada cuando se sabe el número exacto de veces que va a ocurrir un ciclo.

La representamos en el pseudocódigo de la siguiente forma:

para *Variable ← valor_inicial, valor_final, incremento*
hacer *instrucciones*
fin_para

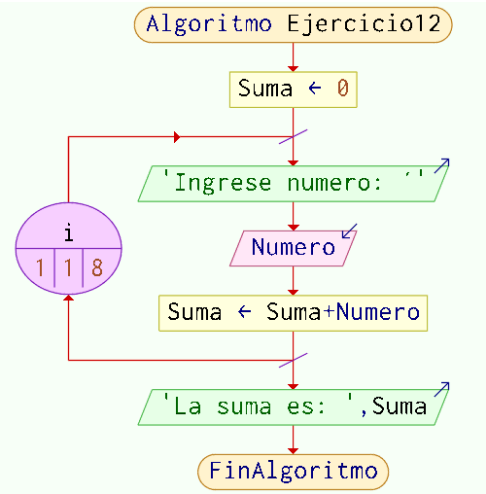
Ejemplo 12: Sumar 8 números introducidos por teclado:

Variables: Suma (Acumulador), Numero, i (Contador que controla el ciclo para)

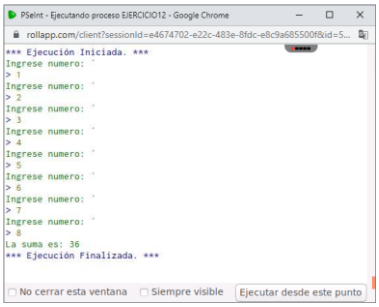
Pseudocódigo

```
Algoritmo Ejercicio12
    Suma <- 0
    Para i <- 1 Hasta 8 Con Paso 1 Hacer
        Escribir 'Ingrese numero: ´'
        Leer Numero
        Suma ← Suma + Numero
    Fin Para
    Escribir 'La suma es: ',Suma
FinAlgoritmo
```

Diagrama de Flujo



Ejecución



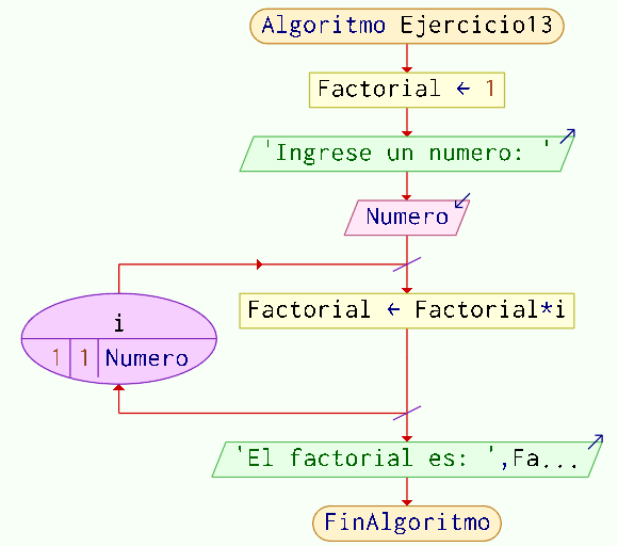
Ejemplo 13: Calcular y escribir el factorial de un número dado por el usuario.

Variables: Numero, Factorial, i (Contador de repetición)

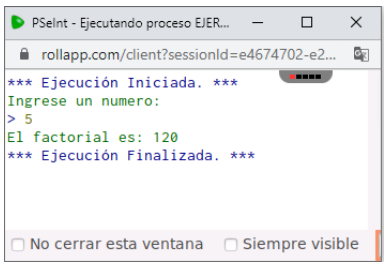
Pseudocódigo

```
Algoritmo Ejercicio13
    Factorial <- 1
    Escribir 'Ingrese un numero: '
    Leer Numero
    Para i <- 1 Hasta Numero Con Paso 1 Hacer
        Factorial <- Factorial * i
    Fin Para
    Escribir 'El factorial es: ',Factorial
FinAlgoritmo
```

Diagrama de Flujo



Ejecución



Ejemplo 14

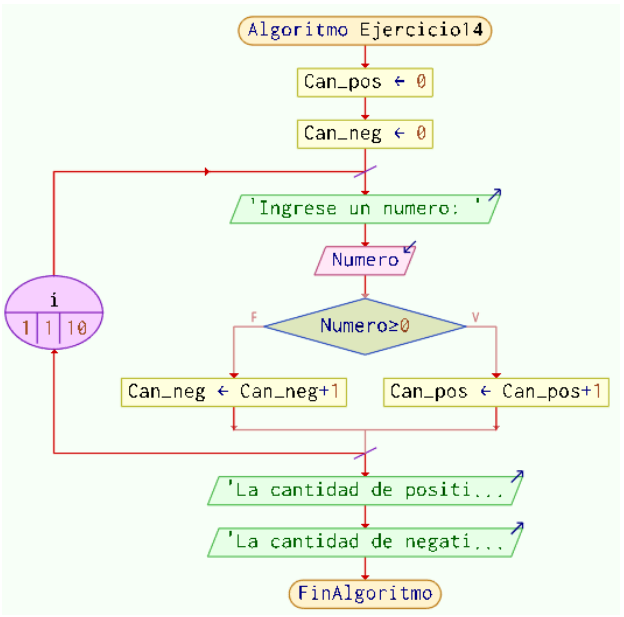
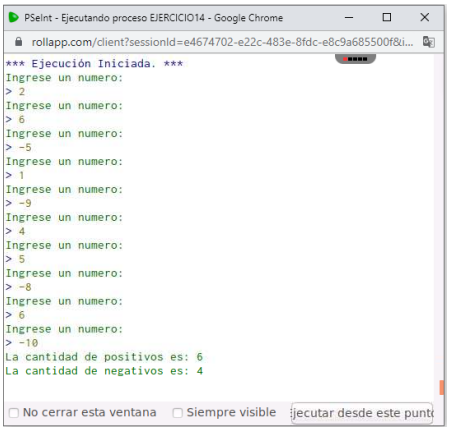
Leer 10 números, calcular y escribir la cantidad de números positivos y la cantidad de números negativos.

Variables: Cant_pos, Cant_neg, Número,

Pseudocódigo

```
Algoritmo Ejercicio14
    Can_pos <- 0
    Can_neg <- 0
    Para i <- 1 Hasta 10 Con Paso 1 Hacer
        Escribir 'Ingrese un numero: '
        Leer Numero
        Si Numero ≥ 0 Entonces
            Can_pos <- Can_pos + 1
        SiNo
            Can_neg <- Can_neg + 1
        Fin Si
    Fin Para
    Escribir 'La cantidad de positivos es: ',Can_pos
    Escribir 'La cantidad de negativos es: ',Can_neg
FinAlgoritmo
```

Diagrama de Flujo



Ejecución